**Test Plan Document: Open Days Tracking Feature in TravelMate**

**1. Introduction**

This test plan defines the strategy, scope, resources, schedule, and types of testing required to validate the Open Days Tracking feature in the TravelMate application. This feature enables Swisscom administrators to track the usage of allocated Open Days, view travel day consumption, and manually reallocate unused days through a new dashboard interface. The goal is to ensure that this feature is implemented correctly, performs reliably, and integrates smoothly into the existing TravelMate infrastructure.

**2. Objectives and Tasks**

**Objectives:**

* Verify that Open Days are accurately tracked and displayed.
* Validate the backend logic for determining Open Day usage based on employee bookings.
* Ensure that admins can manually reallocate unused Open Days.
* Confirm that the frontend dashboard is performant and intuitive.
* Ensure the backend integrates well with internal services and handles failures robustly.

**Tasks:**

* Design and execute unit and integration tests on the backend.
* Develop component and end-to-end (E2E) tests on the frontend.
* Test UI workflows for loading, reallocation, and manual updates.
* Validate logging, security, and error handling.
* Perform regression testing on related TravelMate features.

**3. Scope**

**In Scope:**

* Open Day status tracking.
* Admin-facing dashboard UI.
* Reallocation functionality.
* Internal backend services and APIs.

**Out of Scope:**

* Direct integration testing with AnyOrg.
* Forecasting or automated policy enforcement.

**4. Types of Testing**

|  |  |  |
| --- | --- | --- |
| **Layer** | **Type of Test** | **Description** |
| Backend | Unit Testing | Test isolated business logic (e.g., status determination, booking matching). |
| Backend | Integration Testing | Test REST API endpoints, database interactions, and internal service coordination. |
| Frontend | Component Testing | Test UI components such as status badges, reallocation buttons, and Open Day rows. |
| Frontend | End-to-End Testing | Simulate admin flows such as viewing the dashboard, reallocating a day, or handling errors. |

**5. Test Items**

* REST API endpoints for retrieving and updating Open Day data.
* Open Day dashboard components (frontend).
* Reallocation workflows.
* Logging and audit trails.

**6. Functional Test Cases**

**Backend (Unit + Integration)**

* Validate date-based status assignment (upcoming/passed).
* Match booking records to Open Days.
* Date is no in database when unselected.
* Handle missing or malformed data.

**Frontend (Component + E2E)**

* Render dashboard with correct status indicators.
* Display error messages on failed data fetch.
* Allow admin to manually reallocate a day.
* Show reallocated days as available.
* Prevent unauthorized actions via role checks.

**7. Non-Functional Requirements**

* **Performance:** Dashboard loads within 2 seconds (95% of cases).
* **Availability:** System uptime of 99.5%.
* **Security:** Only authorized admins can change Open Day data.
* **Maintainability:** 80% unit test coverage minimum.
* **Scalability:** Handle 10,000+ requests cross tribes.

**Non-Functional Testing Approach:**

* **Performance:**
  + Use browser dev tools or Lighthouse to measure load times.
  + Run automated E2E tests (e.g., Cypress/Playwright) to measure dashboard render time.
  + Simulate load using JMeter or Locust to stress test backend APIs.
* **Availability:**
  + Monitor system uptime with tools like Prometheus, Grafana, or Pingdom.
  + Test failure recovery and service restarts in a staging environment.
  + Apply chaos engineering techniques to evaluate system resilience.
* **Security:**
  + Implement role-based access tests using E2E tests.
  + Ensure sensitive actions are logged and protected with proper authorization layers.
* **Maintainability:**
  + Check test coverage, using build in IntelliJ tool.
  + Review modularity and code isolation in PRs.
  + Set automated test coverage thresholds in pipelines.
* **Scalability:**
  + Create test datasets with 10,000+ Open Days.
  + Use performance profiling tools (e.g., YourKit, Chrome Profiler) to detect slow queries.
  + Run load and performance tests with large data volumes and concurrent requests.

**8. Test Environment**

* **Frontend:** Angular + Company Component Library (Local + CI pipeline).
* **Backend:** Spring Boot REST services (Dev + Test Environments).
* **Database:** MongoDB with data (Which contains the allowedTravelDates)
* **Auth:** Swisscom internal authentication (SSO).

**9. Test Data**

* Sample tribes with various booking configurations.
* Valid and invalid employee entries.

**10. Exit Criteria**

* All critical tests passed.
* 95% functional coverage with no high-severity bugs.
* Stakeholder approval of dashboard and functionality.

**11. Risks & Mitigations**

|  |  |
| --- | --- |
| **Risk** | **Mitigation** |
| No E2E tests have been written yet | Prioritize E2E test coverage using Cypress or Playwright for all critical user flows. |
| Integration tests are not set up yet | Establish test framework using Spring Boot Test or Testcontainers; add integration tests to CI pipeline. |
| Monitoring is not working properly | The team has been having trouble with the monitoring, this could jeopardize the availability requirement testability. |